WYSIWYN: Using Task Focus to Ease Collaboration

Mik Kersten, Rob Elves and Gail C. Murphy
Department of Computer Science
University of British Columbia
{beatmik, relves, murphy}@cs.ubc.ca

ABSTRACT

WYSIWYG (What You See Is What You Get) tools changed how knowledge workers and others produce and collaborate on documents. Our Mylar project is showing how WYSIWYN[1](What You See Is What You Need) tools can change how programmers work and interact. As a programmer works on a task, Mylar builds a context for the task that captures which resources are interesting to complete the task. These task contexts can be used to focus the user interface with which a programmer works, reducing the overload of information the programmer usually experiences. Sharing task contexts can also focus collaborative programming activities, making it easy to show a team member how a bug was solved. We have shown that Mylar makes programmers more productive in a field study of 16 programmers using Mylar for several weeks. In this position paper, we provide an overview of Mylar and discuss some further ways in which WYSIWYN may improve programmers' use of integrated development environments.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments — *integrated environments, programmer workbench*.

General Terms

Design, Human Factors

Keywords

Task-based interaction, Degree-of-interest, Focused user interfaces

1. INTRODUCTION

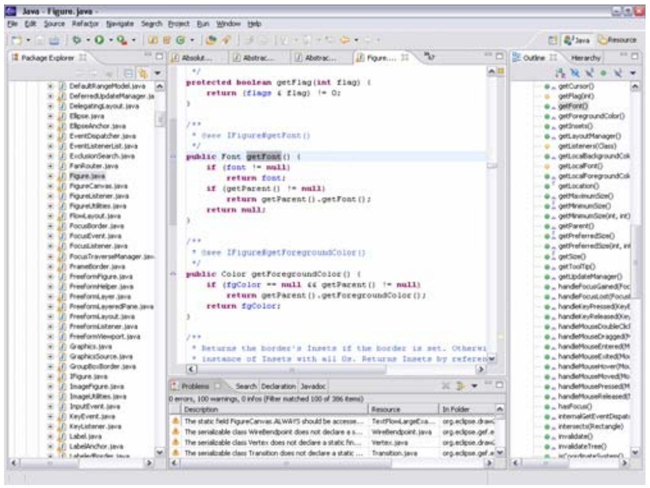
WYSIWYG (What You See Is What You Get) had a fundamental impact on the productivity of knowledge workers. For example, these tools made it possible for organizations to create professional quality newsletters that keep current and past members of an organization aware of events happening at the organization. As another example, these tools changed the work performed by administrative staff as individuals within an organization became responsible for formatting letters and documents they wrote.

In our Mylar project[2], we are investigating how WYSIWYN (What You See Is What You *Need*) tooling can change how individual programmers work and how those programmers work together. WYSIWYN tooling *focuses* the information presented to programmers on just the information that he or she needs to complete individual tasks and to collaborate with others on the tasks associated with a project.

In this position paper, we provide a brief overview of Mylar, describing how it provides WYSIWYN support to both an individual programmer and to teams of programmers. We also briefly discuss further ways in which WYSIWYN support could be added to Integrated Development Environments (IDEs) to facilitate collaboration.

2. MYLAR

Many programmers spend much of their time working in an IDE. The trend in IDEs has been to add more and more features that are able to quickly display more and more information about the system to the programmers. Figure 1 shows a screenshot of the typical views facing a Java programmer working in the Eclipse IDE[3]: each view is populated with numerous program elements, requiring the developer to scroll and search to find the elements needed for the task-at-hand. The result of making it possible to easily display a large subset of a system's artifacts is that programmers spend more time looking for elements they need to complete a task than they spend actually working with those elements. Unless they are systematic in looking for the elements of interest, they can suffer from inattention blindness, missing relevant items that may appear on the screen accidentally [6]. This problem is exacerbated by two aspects of a programmer's work: 1) a programmer switches between tasks frequently [2] and 2) a programmer often collaborates with other team members.



```
    */
    protected boolean getFlag(int flag) {
        return (flags & flag) != 0;
    }

    /**
     * @see IFigure#getFont()
     */
    public Font getFont() {
        if (font != null)
            return font;
        if (getParent() != null)
            return getParent().getFont();
        return null;
    }

    /**
     * @see IFigure#getForegroundColor()
     */
    public Color getForegroundColor() {
        if (fgColor == null && getParent() != null)
            return getParent().getForegroundColor();
        return fgColor;
    }

    /**
     * Returns the border's Insets if the border is set. Otherwi
     * instance of Insets with all 0s. Returns Insets by referen
```

0 errors, 100 warnings, 0 infos (Filter matched 100 of 386 items)

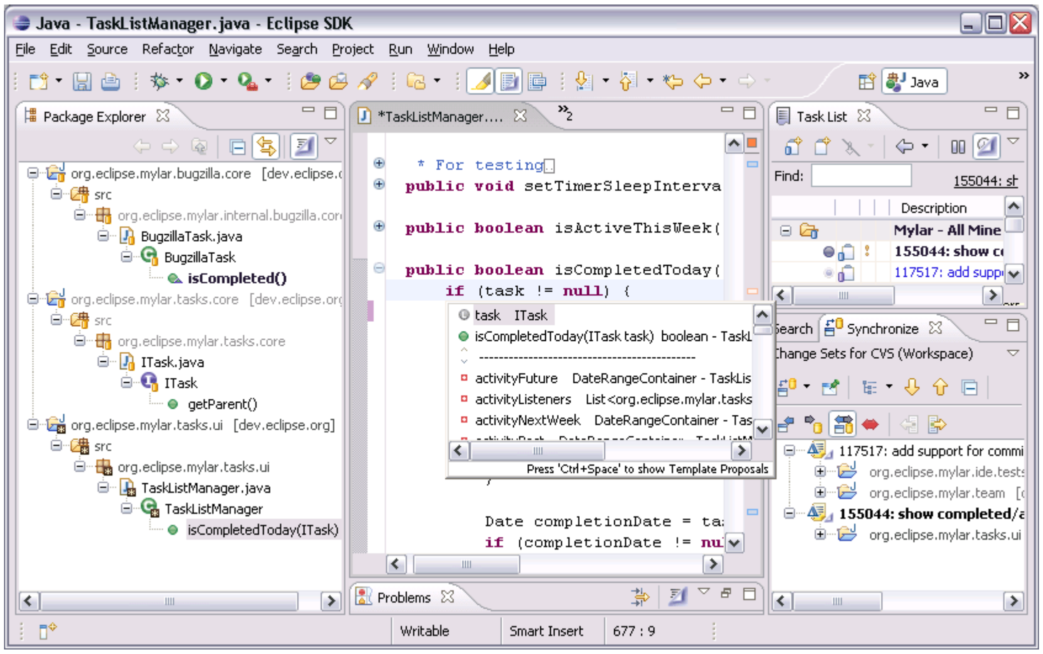
Description	Resource	In Folder
The static field FigureCanvas.ALWAYS should be accesse...	TextFlowLargeExa...	org.eclipse.draw
The serializable class WireBendpoint does not declare a s...	WireBendpoint.java	org.eclipse.gef.e
The serializable class Vertex does not declare a static fin...	Vertex.java	org.eclipse.draw
The serializable class Transition does not declare a static ...	Transition.java	org.eclipse.gef.e

Figure 1 An Overloaded IDE Workspace

[1] www.usabilityfirst.com/glossary/term_1070.txl, verified on 06/10/06

[2] `www.eclipse.org/mylar`, verified on 12/09/06.

[3] `www.eclipse.org`, verified on 12/09/06.


Java - TaskListManager.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
org.eclipse.mylar.bugzilla.core [dev.eclipse.
src
org.eclipse.mylar.internal.bugzilla.core
BugzillaTask.java
BugzillaTask
isCompleted()
org.eclipse.mylar.tasks.core [dev.eclipse.org
src
org.eclipse.mylar.tasks.core
ITask.java
ITask
getParent()
org.eclipse.mylar.tasks.ui [dev.eclipse.org]
src
org.eclipse.mylar.tasks.ui
TaskListManager.java
TaskListManager
isCompletedToday(ITask)
*TaskListManager....
Task List
Find:
155044: sh
Description
Mylar - All Mine
155044: show co
117517: add supp
Search
Synchronize
Change Sets for CVS (Workspace)
117517: add support for commi
org.eclipse.mylar.ide.tests
org.eclipse.mylar.team
155044: show completed/a
org.eclipse.mylar.tasks.ui
Problems
Writable
Smart Insert
677 : 9

```
 * For testing
public void setTimerSleepInterva

public boolean isActiveThisWeek(

public boolean isCompletedToday(
    if (task != null) {

    Date completionDate = ta
    if (completionDate != nu
```

```
task  ITask
isCompletedToday(ITask task) boolean - TaskL
------------------------------------------
activityFuture  DateRangeContainer - TaskLis
activityListeners  List<org.eclipse.mylar.tasks
activityNextWeek  DateRangeContainer - Tas
Press 'Ctrl+Space' to show Template Proposals
```

Figure 2: A Task-Focused Workspace with Mylar

Our Mylar tool addresses the overload problem by focusing the information presented in the IDE around tasks. For instance, a programmer can see just the information needed to work on a particular task. One team member can see the status of tasks (issues) being worked on within the team. When one programmer collaborates with another, the first can easily share just the information that he or she considered and changed while working on the task with the team member.

2.1 Mylar for an individual programmer

A programmer working with Mylar indicates the current task by indicating the issue on which they are working from an issue repository.[4] Mylar's Task List supports queries over an issue repository. The Task List in the upper right view of Figure 2 shows tasks resulting from a query over a Bugzilla repository[5]. To indicate a particular issue is the current task, the programmer activates the issue by pressing the small round button at the left of a particular issue's name. In this case, the programmer has activated the issue #155044.

Once a task is activated, Mylar begins to monitor the resources (program elements and other information) a programmer accesses to perform the task. With this information, Mylar builds a model of which resources are important for the task. This model assigns a degree-of-interest to each resource based on the number of edits and selections of the resource [3,4]. We refer to the degree-of-interest model for a task as a task context. A task's context can be used as input to several operations. For instance, the context can be used to highlight the information presented or it can be used to filter uninteresting information. The left side of Figure 2 shows the Package Explorer (a view displaying the hierarchical containment hierarchy of the software) filtered to show only resources interesting for the current task. Comparing this to the workspace shown in Figure 1, we see that activation of the task focuses the views in the IDE to just what the programmer needs at present. Mylar retains the context for a task between activations of the task. When a programmer returns to work on a task, the programmer simply needs to reactivate the task and Mylar reloads the context displaying only the information needed for the task.

To investigate whether Mylar does enable programmers to spend more time working with resources than looking for them, we performed a field study in which 16 programmers used Mylar for their daily work for a period of several weeks. We benchmarked the activity of these programmers in Eclipse prior to providing them with Mylar. With statistical significance, we found that these programmers spent more time editing code than navigating it when using Mylar [4].

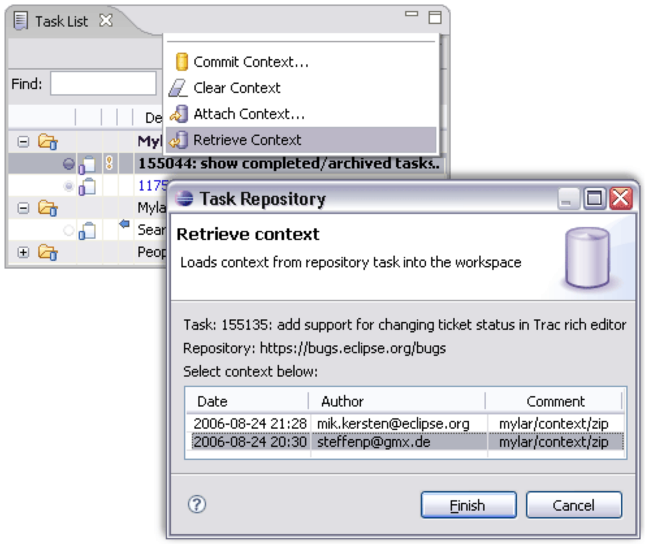
2.2 Mylar for collaborating programmers

Two or more programmers often end up working on the same task. This work may occur by the programmers huddling around the same workspace, it may occur by the programmers sequentially passing the task back and forth with one programmer making some progress and then passing it to another who has expertise in a different area, or it may occur separated in time with one programmer revisiting a previously completed task because of a newly reported bug or a desired enhancement.

Mylar provides assistance to programmers in each of these scenarios. When multiple programmers work on a task simultaneously at one computer, the focus provided by the task context can make it easier for the multiple programmers to discuss the software and for the non-driving programmers to follow the actions of the driving programmer on the screen.

[4] Mylar also supports individual tasks known only to the programmer. In this paper, we will note significant features that we do not have room to discuss; for more details on any of these features and others, see the Mylar website (`www.eclipse.org/mylar`).

[5] `www.bugzilla.org`, verified 15/09/06



155044: show completed/archived tasks..

Task Repository

Retrieve context

Loads context from repository task into the workspace

Task: 155135: add support for changing ticket status in Trac rich editor

Repository: https://bugs.eclipse.org/bugs

Select context below:

Date	Author	Comment
2006-08-24 21:28	mik.kersten@eclipse.org	mylar/context/zip
2006-08-24 20:30	steffenp@gmx.de	mylar/context/zip

Figure 3 Context Sharing Initiated from the Task List

Mylar provides assistance for the other two scenarios by facilitating the sharing of task contexts. We have experimented with two means of sharing such contexts. First, most commonly, we attach the context used in solving a problem to the report describing the problem in a shared issue repository. Second, a context for a task can be exchanged in email. As Mylar is an open-source project, we prefer the first approach to provide transparency in the development process.

A programmer who wishes to work on a task left off by another programmer or who wishes to see how an issue was resolved can import the context for that task into their workspace (Figure 3). Similar to how one programmer can switch between tasks, a programmer importing a task context can switch to that context thereby accessing just that subset of the software system the original programmer had considered in completing the task. When contexts for tasks are stored in a shared repository, such as the Bugzilla repository used for the Mylar development itself, the repository becomes a richer source of knowledge about how to complete problems. Currently, the Mylar repository has contexts attached for 253 tasks (as of September 14, 2006). These shared contexts have made it much easier for Mylar's developers to work on reopened bug reports and to delegate partially completed tasks. The Mylar project also has a policy of attaching task contexts with submitted patches. This policy has made it much easier to apply dozens of contributed patches each month.

Mylar also helps focus communication between multiple programmers by providing a rich, focused interaction with a shared issue repository. In particular, Mylar supports the use of queries to watch for changes in particular categories of issues. For example, a query may be used to watch all updates made to an issue by a particular colleague. When the colleague adds a comment to an issue, the programmer will see the issue appear under the query in the Mylar Task List and will see an incoming arrow to represent changes have been made in the repository to the issue. When the programmer opens the issue, the view of the issue reflects the latest changes; for instance, except for the latest unseen comments, conversations are folded.

3. DISCUSSION

Another way to add more WYSIWYN support to IDEs is to provide recommendations. Mylar provides an experimental form of recommendation called Context Search, which for particularly interesting resources automatically runs and displays reference-based searches. For example, the Context Search may display the callers of a particular method that has a high-degree of interest. By knowing the context of a task, in a similar way, when a programmer begins work on a new problem report, it is possible to recommend previous problems completed in the past and to provide rich support in suggesting which parts of the system may be relevant to solving the problem [5]. These recommendations provide focus when they are sufficiently accurate; inaccurate recommendations would reduce the focus of the programmer.

At times, it would also be useful for collaboration to know which parts of the system are concurrently being worked on by other team members. One way of presenting this information is through decorations to resources in the IDE [1]. This decoration can be overwhelming when applied to all resources in the system. By knowing which resources programmers are working on and what they are doing with those resources, it may be possible to use task context to scope the collaboration information presented to provide more focus. For example, dynamically determining who the current team is working on a task and focusing collaboration affordances in the UI around that team.

4. SUMMARY

Most programmers' work is structured by the tasks that they perform. Mylar uses information gathered about how programmers work on tasks to focus the display of system information to the programmer and to focus the interaction of the programmer with the development environment. In this position paper, we have provided a brief overview of some of the facilities provided by Mylar to focus individual and team programming efforts. A full description of Mylar's features can be found at the project's website. Mylar ensures that what the programmer needs is what the programmer gets.

5. ACKNOWLEDGEMENTS

This work has been funded by NSERC and IBM. We are grateful to the many users of Mylar that have greatly enhanced its functionality.

6. REFERENCES

[1] Cheng, L., Hupfer, Susanne, Ross, Steven and Patterson, John. Jazzing up Eclipse with Collaborative Tools. *Proc. of the 2003 OOPSLA Workshop on Eclipse Technology Exchange*, pp. 45-49.

[2] Gonzales, V.M. and Mark G. Constant, constant, multi-tasking craziness: managing multiple working spheres. *Proc. of the Conf. on Human Factors in Computing Systems*, 2004, pp. 113-120.

[3] Kersten, M. and Murphy, G.C. Mylar: a degree-of-interest model for IDEs. *Proc. of the Conf. on Aspect-oriented Software Development*, 2005, pp. 159-168.

[4] Kersten, M. and Murphy, G.C. Using task context to improve programmer productivity. To appear, *Proc. of Conf. on Foundations of Software Engineering*, 2006.

[5] Murphy, G.C., Kersten, M., Robillard, M.P. and Čubranić. D. The emergent structure of development tasks. *Proc. of European Conf. on Object-oriented Programming*, 2005, pp. 33-48.

[6] Robillard, M.P., Murphy, G.C., and Coelho, W. How effective developers investigate source code: an exploratory study. *IEEE Transactions on Software Engineering*, Vol 30, No 12, 2004, pp. 889-903.